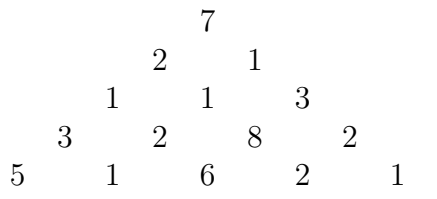
Informatique du Tronc Commun

TD n°3 Programmation dynamique

Exercice n°1 Coût d'un chemin



On considère une pyramide de nombres comme celle de l'exemple ci-contre. En partant de celui du haut (ici 7), le but est de trouver le chemin vers le bas qui maximise la somme de tous les nombres parcourus. Chaque entier peut être suivi d'une des deux cases inférieures.

Dans cet exemple, on peut espérer obtenir : $25 = 7 + 1 + 3 + 8 + 6$

Pour représenter la pyramide précédente, on utilise une liste de listes :

```
P = [
    [7],
    [2, 1],
    [1, 1, 3],
    [3, 2, 8, 2],
    [5, 1, 6, 2, 1]
    ]
```

On peut aussi créer une pyramide aléatoire de `n` lignes, avec la commande :

```
from random import randint
P = [ [ randint (0,9)  for  j  in range(i) ] for i in range(1,n+1) ]
```

On cherche à écrire une fonction prenant comme argument la liste `P`, une ligne `i` et une colonne `j`, et renvoyant la somme maximale vers le bas de la pyramide à partir du nombre d'indices `i` et `j`.

Q1. Questions préliminaires sur la liste de listes `P`.

(a) Quelle valeur renvoie `P[3][1]` ?

(b) Quels sont les nombres situés en dessous de `P[i][j]` ?

Q2. Que doit renvoyer la fonction dans le cas où `i` correspond à la dernière ligne ?

Q3. À partir de la case `(i,j)`, quelles sont les deux possibilités à envisager ? Laquelle faut-il choisir ? Exprimer récursivement ce que doit renvoyer la fonction.

Q4. Écrire la fonction récursive `somme_rec(P,i,j)` qui renvoie la la somme maximale vers le bas de la pyramide à partir du nombre d'indices `i` et `j`.

Q5. Comment obtenir la somme maximale de la pyramide ?

Q6. Évaluer la complexité de la fonction précédente.

On cherche à présent à mettre en place une programmation dynamique. Pour cela, il faudra mémoïser les résultats intermédiaires à l'aide d'un dictionnaire dont la clé est un 2-uplet des deux indices du nombre correspondant et la valeur la somme maximale à partir de ce nombre.

Q7. Modifier la fonction précédente afin d'écrire sa version dynamique `somme_mem(P,i,j,dico={})`.

Exercice n°2 Plus longue séquence commune

La distance d'édition permet de mesurer le degré de similarité de deux chaînes. Elle ne donne pas d'information quant aux séquences maximales communes aux deux chaînes. Or, en génétique par exemple, il peut s'avérer très important de savoir quels sont les points communs de deux génomes. Le problème de la plus longue sous-chaîne permet d'apporter une réponse à cette question.

Définition : Sous-chaine d'une chaîne de caractères

On appelle sous-chaine d'une chaîne de caractères $a = a_0...a_{n-1}$ toute chaine de caractères s extraite de a telle que $s = a_i...a_k$ où $0 \leq i... \leq k \leq n-1$ et $\forall p \in \{i, ..., k\}, a_p \in$. Les caractères de s n'apparaissent pas nécessairement de manière consécutive dans la chaîne a.

Définition : Plus longue sous-chaine commune

Soit $a = a_1...a_q$ et $B = b_1...b_p$ deux chaines de caractères non vides. On appelle plus longue sous-séquence commune à a et b toute sous-chaîne commune à a et b de longueur maximale.

Si l'une des chaînes a ou b est vide ou si a et b n'ont aucune sous-chaîne commune, la chaine vide est alors l'unique plus longue sous-chaîne commune à a et b.

On note $\mathcal{L}(a, b)$ la fonction qui renvoie la longueur maximale d'une sous-chaîne commune à a et b.

Q1. On considère les chaines `a="AATGCG"` et `b="TATTAGC"`, identifier la (les) plus longues sous-chaînes communes et donner la valeur de $\mathcal{L}(a, b)$.

Soient les deux chaines de caractères a et b de longueurs n_a et n_b.

Soit $\ell(i, j)$ la longueur de la plus longue sous-séquence des chaines extraites $a_0a_1...a_{i-1}$ et $b_0b_1...b_{j-1}$ de longueurs i et j, des chaînes a et b, de longueurs n_a et n_b.

Q2. On cherche à établir la relation de récurrence permettant d'exprimer $\ell(i, j)$.

(a) Si l'une des deux chaînes extraites est vide (c'est-à-dire $i = 0$ ou $j = 0$), que vaut $\ell(i, j)$?

(b) Si $a_{i-1} = b_{j-1}$, par quel caractère termine la plus longue sous-séquence commune ?
Quel est le lien entre la longueur de la plus longue sous-séquence commune de $a_0...a_{i-1}$ et $b_0...b_{j-1}$, et celle entre les chaînes privées de leur dernier caractère (a_{i-1} et b_{j-1}).
En déduire l'expression de $\ell(i, j)$ en fonction de $\ell(i-1, j-1)$.

(c) Si $a_{i-1} \neq b_{j-1}$, il faut chercher la plus longue sous-séquence commune entre la chaine extraite de a privée de a_{i-1} et b, et le cas inverse.
Exprimer $\ell(i, j)$ en fonction de $\ell(i-1, j)$ et $\ell(i, j-1)$.

(d) Compléter la relation de récurrence ci-dessous :

$$\forall i \in [\![0, n_a]\!], \forall j \in [\![0, n_b]\!], \ell(i, j) = \begin{cases} ..... & \text{si } i = 0 \text{ ou } j = 0 \\ ..... & \text{si } a_{i-1} = b_{j-1} \\ .....\big(\ell(....., ...), \ell(....., ...)\big) & \text{si } a_{i-1} \neq b_{j-1} \end{cases}$$

Q3. Pour la résolution de bas en haut, on utilise un tableau dans lequel on stocke les résultats au fur et à mesure, en partant de la plus petite chaine possible (la chaine vide) en allant jusqu'au problème souhaité.

On s'intéresse à la recherche de la longueur de la plus longue sous-séquence commune entre `a="AATGCG"` et `b="TATTAGC"`.
Pour cela recopier et remplir le tableau ci-contre, qui contient dans la case (i, j) la longueur de la plus longue sous-séquence commune à la chaine `a[:i]` (les i premiers caractères de a) et à la chaine `b[:j]` (les j premiers caractères de b).

i \ j	0	1	2	3	4	5	6	7
0								
1								
2								
3								
4								
5								
6								

Q4. Compléter la fonction `L_bh(ch1:str,ch2:str)->int` qui résout $\mathcal{L}(ch1, ch2)$ avec la programmation dynamique de bas en haut, c'est-à-dire qui renvoie la plus longue sous-séquence commune aux deux chaines de caractères `ch1` et `ch2`.

ATTENTION il y a un décalage de 1 entre la place dans le tableau et la place du caractère dans la chaîne de caractères, quand on remplit la case `tab[i][j]` on compare les caractères situés, dans les chaines de caractères, aux places `i-1` et `j-1`.

```
def L_bh(ch1:str,ch2:str)->int:
    n1,n2=len(ch1),len(ch2)
    # initialisation du tableau avec des 0
    tab=
    for i in range(1,n1+1):
        for j in range(1,n2+1):
            if                                    : # le dernier caractère
  de ch1[:i] est identique au dernier caractère de ch2[:j]
                tab[i][j]=
            else:
                a=                              # longueur de la plus longue
  chaine commune à ch1[:i-1] et à ch2[:j]
                b=                              # longueur de la plus longue
  chaine commune à ch1[:i-1] et à ch2[:j]
                tab[i][j]=                      # on choisit la plus longue
  des deux précédentes
    return                        # dernière case du tableau
```

Q5. Compléter la fonction `L_mem(ch1:str,ch2:str,d={})->int` ci-dessous qui résout $\mathcal{L}(ch1, ch2)$ récursivement, donc de haut en bas, avec mémoïsation.

On stocke les calculs successifs dans le dictionnaire `d` dont la clé est un 2-uplet des deux chaines considérées, et la valeur associée est la longueur de la plus longue sous chaine commune à ces deux chaines.

```
def L_mem(ch1:str,ch2:str,d={})->int:
    if (ch1,ch2) in d: # le calcul a déjà été fait pour ces deux chaines

    else:
        if                  : # si l'une des chaines est vide
            c=
        elif                : # si les deux derniers caractères sont
  identiques
            c=
        else:
            a=               # longueur de la plus longue sous-séquence
  commune à ch1 privée du dernier caractère, et à ch2
            b=               # longueur de la plus longue sous-séquence
  commune à ch2 privée du dernier caractère, et à ch1
            c=               # max des deux longueurs précédentes
        d[(ch1,ch2)]=       # on enregistre la valeur calculée
        return dico[(ch1,ch2)]
```

Exercice n°3 Partition équilibrée d'un tableau d'entiers positifs

Vous partez en randonnée avec un.e ami.e. Vous avez un certain nombre, noté n, de choses à emporter : tente, victuailles, duvets, ... Vous souhaitez répartir toutes les choses à porter sur vos deux dos de la façon la plus équitable possible. Nous cherchons un algorithme pour répartir ces choses de façon optimale.
On note $C = \{i\}_{i\in[\![0,n-1]\!]}$ l'ensemble des choses à emporter, chacune de masse en gramme, m_i, avec $m_i \in \mathbb{N}$.
On note C_1 l'ensemble des choses que vous allez devoir porter, de masse $M_1 = \sum_{k\in C_1} m_k$, et C_2 l'ensemble des choses que votre ami.e va devoir porter, de masse $M_2 = \sum_{k\in M_2} m_k$.
Ainsi, on cherche C_1 et C_2 tel que $C_1 \cup C_2 = C$, avec $C_1 \cap C_2 = \emptyset$, en minimisant $|M_1 - M_2|$.

Q1. Quelle est la valeur de M_1 et M_2 qui minimise $|M_1 - M_2|$ (c'est-à-dire égal à 0, même si ce n'est pas atteignable) ? On note cette valeur M_I.

Q2. Combien de partitions faudrait-il évaluer si l'on voulait trouver la partition optimale en les énumérant toutes ? Commenter.

Q3. Soit $P(i, m)$ une fonction booléenne qui indique s'il existe un sous-ensemble des i premières choses, $\{0, 1, \ldots, i-1\}$, dont la masse totale est exactement m. Compléter la définition de $P(i, m)$ par récurrence ci-dessous.

$P(i,m) = \begin{cases}$

				True / False
si	$i = 0$	et	$m = 0$	
si	$i = 0$	et	$m \neq 0$	
si	$P(i-1, m)$	$=$	$True$	
si	$P(i-1, m-m_{i-1})$	$=$	$True$	

On envisage une approche ascendante, pour laquelle on stocke les résultats successifs en commençant par les plus petits sous-problèmes.

Q4. Compléter le tableau ci-dessous contenant les valeurs de $P(i, m)$ pour toutes les valeurs de $i \in [\![0, n]\!]$ et $m \in [\![0, M]\!]$ pour l'ensemble des masses suivantes : $m_0 = 8$, $m_1 = 5$, $m_2 = 2$, $m_3 = 2$ ($n = 4$). La masse totale est donc $M = 17$.

Chaque ligne du tableau correspond à une valeur de i et on rappelle la valeur des masses du sous-ensemble $\{0, 1, ..., i-1\}$. Chaque colonne correspond à une masse m de $[\![0, M]\!]$.

On mettra un « T » pour True, et on pourra se contenter d'un « · » pour False.

i \ m	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
$i = 0$ $\emptyset$																		
$i = 1$ $m_0 = 8$																		
$i = 2$ $m_1 = 5$																		
$i = 3$ $m_2 = 2$																		
$i = 4$ $m_3 = 2$																		

Q5. Quelle est la complexité d'un algorithme qui compléterait un tableau donnant toutes les valeurs $P(i, m)$?

Q6. À partir de P et M_I, comment peut-on donner la valeur minimale de $|M_1 - M_2|$?
Calculer la valeur minimale de $|M_1 - M_2|$ pour les masses $\{8, 5, 2, 2\}$.

Q7. Expliquer à l'aide du tableau P comment trouver un groupe qui s'approche le plus du poids idéal ?

Q8. Écrire la fonction `tableau(C:list)->list` qui à partir de la liste des masses des objets renvoie le tableau des valeurs (`True` ou `False`) de P.
On fera attention au décalage de 1 entre le rang dans la liste `C` et le numéro de la ligne.

Q9. Écrire la fonction `partition(C:list)->(list,list)` qui à partir du tableau des valeurs de P renvoie les deux listes C_1 et C_2 des ensembles de choses à emporter tels que $|M_1 - M_2|$ soit minimal. Pour cela, il faut :

- Remplir le tableau des valeurs de P,
- Déterminer la masse atteignable la plus proche de $MI = M/2$. C'est M_I si dans le tableau dans la colonne `MI` ligne `n` il y a `True`. Sinon il faut la calculer : utiliser Q6.
- Utiliser Q7 pour finir.